@git.zone/tsrust

Documentation for @git.zone/tsrust

- readme.md for @git.zone/tsrust
- changelog.md for @git.zone/tsrust

readme.md for @git.zone/tsrust

A CLI build tool for Rust projects that follows the same conventions as `@git.zone/tsbuild`. It detects your `rust/` source directory, parses `Cargo.toml` (including workspaces), runs `cargo build --release`, and copies the resulting binaries into a clean `dist_rust/` directory at the project root.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

Install globally via npm:

```
npm install -g @git.zone/tsrust
```

Or as a project-level dev dependency:

```
pnpm install --save-dev @git.zone/tsrust
```

> ⚡ **No Rust required!** If `cargo` isn't found on your system, `tsrust` automatically downloads and installs a minimal Rust toolchain to `/tmp/tsrust_toolchain/`. This gives a zero-setup experience. If you already have Rust installed, `tsrust` uses your system toolchain.

The Convention

`tsrust` mirrors the directory convention established by `tsbuild`:

Tool	Source Directory	Output Directory
`tsbuild`	`ts/`	`dist_ts/`
`tsrust`	**`rust/`**	**`dist_rust/`**

Your Rust code lives in `rust/` (or `ts_rust/` as fallback), and compiled binaries land in `dist_rust/` — ready for packaging, deployment, or further tooling.

Usage

□□ Build (Default Command)

Simply run `tsrust` from your project root:

```
tsrust
```

This will:

1. Detect the Rust toolchain (system `cargo`, or auto-install to `/tmp/tsrust_toolchain/`)
2. Locate your `rust/` directory (containing `Cargo.toml`)
3. Parse the workspace to discover all `[[bin]]` targets
4. Run `cargo build --release` with full streaming output
5. Copy each binary to `dist_rust/` with executable permissions (`chmod 755`)
6. Report file sizes and total build time

Example output:

```
Using cargo 1.90.0 (840b83a10 2025-07-30)
Found Rust project at: rust
Detected Cargo workspace
Binary targets: rustproxy
Running: cargo build --release
   Compiling rustproxy v0.1.0
    Finished `release` profile [optimized] target(s) in 29.01s
Copied rustproxy (13.4 MB) -> dist_rust/rustproxy
Done in 29.2s
```

Automatic Rust Toolchain

`tsrust` provides a zero-setup experience through automatic toolchain management:

1. **System toolchain detected** → uses it as-is (no download, no overhead)
2. **No system toolchain** → checks `/tmp/tsrust_toolchain/` for a previously installed bundled toolchain
3. **No bundled toolchain** → downloads `rustup-init` and installs a minimal Rust toolchain (~70-90 MB download) to `/tmp/tsrust_toolchain/`

The bundled toolchain is stored in `/tmp/`, so it's cleaned up on reboot. Subsequent runs reuse the existing installation. This is ideal for CI environments and quick starts where you don't want to manage a system-wide Rust install.

Supported platforms for automatic install: Linux (x64, arm64) and macOS (x64, arm64).

□□ Debug Build

Build with the debug profile instead of release:

```
tsrust --debug
```

Binaries are taken from `rust/target/debug/` instead of `rust/target/release/`.

□□ Clean Before Building

Run `cargo clean` before building to force a full rebuild:

```
tsrust --clean
```

Cross-Compilation

Cross-compile for different OS/architecture combinations using the `--target` flag:

```
# Cross-compile for a single target
tsrust --target linux_arm64

# Cross-compile for multiple targets
tsrust --target linux_arm64 --target linux_amd64
```

```
# Full Rust triples are also accepted
tsrust --target aarch64-unknown-linux-gnu
```

Supported friendly target names:

Friendly name	Rust target triple
`linux_amd64`	`x86_64-unknown-linux-gnu`
`linux_arm64`	`aarch64-unknown-linux-gnu`
`linux_amd64_musl`	`x86_64-unknown-linux-musl`
`linux_arm64_musl`	`aarch64-unknown-linux-musl`
`macos_amd64`	`x86_64-apple-darwin`
`macos_arm64`	`aarch64-apple-darwin`

When using `--target`, output binaries are named `<binname>_<os>_<arch>`:

```
dist_rust/
├── rustproxy_linux_arm64
└── rustproxy_linux_amd64
```

`tsrust` automatically installs missing rustup targets via `rustup target add` when needed.

Configuration via .smartconfig.json

You can set default cross-compilation targets in your project's `.smartconfig.json` file so you don't need to pass `--target` flags every time:

```
{
  "@git.zone/tsrust": {
    "targets": ["linux_arm64", "linux_amd64"]
  }
}
```

When targets are configured in `.smartconfig.json`, simply running `tsrust` will cross-compile for all listed targets. CLI `--target` flags take full precedence — if any `--target` is provided, the `.smartconfig.json` targets are ignored entirely.

□□ Clean Only

Remove all build artifacts without rebuilding:

```
tsrust clean
```

This runs `cargo clean` in the Rust directory and deletes the `dist_rust/` output directory.

Project Structure

`tsrust` expects your project to follow this layout:

```
my-project/
├── rust/                   # □□Your Rust source code
│   ├── Cargo.toml          #    Root manifest (workspace or single crate)
│   ├── src/
│   │   └── main.rs         #    (for single-crate projects)
│   └── crates/             #    (for workspace projects)
│       ├── my-binary/
│       │   ├── Cargo.toml  #    Contains [[bin]] targets
│       │   └── src/
│       └── my-lib/
│           ├── Cargo.toml
│           └── src/
├── dist_rust/              # □□Output: compiled binaries go here
│   └── my-binary
├── ts/                     #    (your TypeScript code, built by tsbuild)
├── dist_ts/                #    (TypeScript output)
└── package.json
```

Workspace Support

`tsrust` fully supports Cargo workspaces. It reads the `[workspace]` section from your root `Cargo.toml`, iterates through all `members`, and discovers binary targets from each member crate's `Cargo.toml`.

Binary target discovery follows Cargo's own rules:

- **Explicit `[[bin]]` entries** → uses the `name` field from each entry
- **Implicit binary** → if no `[[bin]]` is declared but `src/main.rs` exists, uses the `[package] name`
- **Library-only crates** → skipped (no binary output expected)

Fallback Directory

If no `rust/` directory is found, `tsrust` checks for `ts_rust/` as a fallback. This supports projects that use the `ts_` prefix convention for all source directories.

Programmatic API

`tsrust` exports its internals for use in other Node.js/TypeScript tools:

```
import { CargoConfig, CargoRunner, FsHelpers, TsRustCli } from '@git.zone/tsrust';

// Parse a Cargo workspace
const config = new CargoConfig('/path/to/rust');
const info = await config.parse();
console.log(info.isWorkspace);   // true
console.log(info.binTargets);    // ['rustproxy']

// Run cargo build
const runner = new CargoRunner('/path/to/rust');
const result = await runner.build({ debug: false, clean: false });
console.log(result.success);     // true
console.log(result.exitCode);    // 0

// File helpers
await FsHelpers.ensureEmptyDir('/path/to/dist_rust');
await FsHelpers.copyFile(src, dest);
await FsHelpers.makeExecutable(dest);
const size = await FsHelpers.getFileSize(dest);
console.log(FsHelpers.formatFileSize(size));  // "13.4 MB"
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the license file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary

use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

changelog.md for @git.zone/tsrust

2026-03-24 - 1.3.2 - fix(config)

migrate project metadata to .smartconfig.json and update related dependencies

- replace npmextra.json with .smartconfig.json in packaged files and documentation
- bump smartconfig, smartshell, smol-toml, tsbuild, tstest, and @types/node versions

2026-03-24 - 1.3.1 - fix(cli)

replace npmextra config loading with smartconfig

- updates the CLI to read tsrust configuration via @push.rocks/smartconfig
- replaces the @push.rocks/npmextra dependency with @push.rocks/smartconfig
- adjusts config fallback documentation to reference smartconfig.json

2026-02-09 - 1.3.0 - feat(toolchain)

add automatic bundled Rust toolchain fallback and integrate with CLI/CargoRunner

- Introduce ToolchainManager to download and install a minimal Rust toolchain to /tmp/tsrust_toolchain/ when system cargo is absent
- Add getEnvPrefix() and installation/verification logic to ToolchainManager; supports Linux and macOS (x64, arm64)
- Make CargoRunner accept an envPrefix and prepend it to cargo/rustup commands so bundled toolchain can be used transparently
- Update CLI to resolve toolchain at runtime (use system cargo if available; otherwise auto-install bundled toolchain) and pass envPrefix to CargoRunner for builds and clean
- Update exports to include mod_toolchain and add new ts/mod_toolchain module files
- Document the automatic toolchain behavior in readme.md and update usage description
- Bump dependencies: @push.rocks/smartcli ^4.0.20 and @types/node ^25.2.2

2026-02-09 - 1.2.0 - feat(cli)

support default cross-compilation targets from npmextra.json

- Add @push.rocks/npmextra dependency and export plugin in ts/plugins.ts
- Introduce ITsrustConfig and read configuration via plugins.npmextra.Npmextra in TsRustCli
- Use npmextra.json targets as fallback when no CLI --target flags are provided
- Update README to document npmextra.json configuration for default targets

2026-02-09 - 1.1.0 - feat(cross-compile)

add cross-compilation support with --target flag, friendly target aliases, and automatic rustup target installation

- tsrust CLI: add --target flag (can be provided multiple times) to cross-compile for specified targets
- Introduce friendly target aliases (linux_amd64, linux_arm64, linux_amd64_musl, linux_arm64_musl, macos_amd64, macos_arm64) and resolve them to Rust triples
- CargoRunner.build accepts a target parameter and CargoRunner.ensureTarget installs missing rustup targets automatically
- Cross-compiled binaries are copied into dist_rust and named _ (e.g., rustproxy_linux_arm64)
- README updated with cross-compilation usage and supported target aliases
- Native build behavior is preserved when --target is not provided

2026-02-09 - 1.0.3 - fix(tsrust)

bump patch version due to no changes

- current package.json version: 1.0.2
- git diff shows no changes; creating a no-op/metadata patch release

2026-02-09 - 1.0.2 - fix()

no changes

- No files changed in the working tree; nothing to commit.

2026-02-09 - 1.0.1 - release

Initial release.

- Initial commit ("initial")
- Project set to version 1.0.1